Tracking Bad Apples: Reporting the Origin of Null & Undefined Value Errors

Michael D. Bond
UT Austin
Nicholas Nethercote
National ICT Australia
Stephen W. Kent
UT Austin
Samuel Z. Guyer
Tufts University
Kathryn S. McKinley
UT Austin

Example Code

User code:

```java
float[][] data = 
    {{1.0f, 2.0f}, {3.0f, 4.0f}};

ScatterPlot plot =
    new ScatterPlot(data, null);
...

plot.draw(...);
```
Example Code

User code:

```java
float[][] data = 
    {{1.0f, 2.0f}, {3.0f, 4.0f}};

ScatterPlot plot =
    new ScatterPlot(data, null);

... plot.draw(...);
```

```
NullPointerException
    at ScatterPlot.draw():315
    at Test.doStuff():124
```

Example Code

User code:

```java
float[][] data = 
    {{1.0f, 2.0f}, {3.0f, 4.0f}};

ScatterPlot plot =
    new ScatterPlot(data, null);

... plot.draw(...);
```

```
NullPointerException
    at ScatterPlot.draw():315
    at Test.doStuff():124
```
### Unusable Value

- Null is **unusable** value
  - Use causes error
- How/why did it become null?
  - Null’s origin?

```java
ScatterPlot {  
  draw(...) {  
    ...  
    xaxis.draw();  
    ...  
  }  
}
```

```
plot.draw(...);
```

```
NullPointerException  
at ScatterPlot.draw():315  
at Test.doStuff():124
```

### Origin of Unusable Value

```java
float[][] data =  
  {{1.0f, 2.0f}, {3.0f, 4.0f}};
```

```java
ScatterPlot plot =  
  new ScatterPlot(data, null);
```

```
plot.draw(...);
```

```
Origin: Test.init():37
```
**Origin of Unusable Value**

```java
float[][] data = 
    {{1.0f, 2.0f}, {3.0f, 4.0f}};

ScatterPlot plot =
    new ScatterPlot(data, null);
...
plot.draw(...);
```

---

**Origin Tracking**

```java
float[][] data = 
    {{1.0f, 2.0f}, {3.0f, 4.0f}};

ScatterPlot plot =
    new ScatterPlot(data, null);
...
```

---

Track every unusable value’s origin for 4% overhead

Key: store origin in place of unusable value
Outline

- Introduction
- Unusable values
- Instances of origin tracking
  - Null pointer exceptions (Java)
  - Undefined value errors (C/C++)
- Redefining program operations
- Evaluation
  - Performance
  - Usefulness
- Related Work

Unusable Values

- Using value causes error
- Examples:
  - Null values
  - Undefined values
- Tough bugs: no info
- **Why** is value unusable?
  - Where did unusable value originate?
Unusable Values

- Using value causes error
- Examples:
  - Null values
  - Undefined values
- Tough bugs: no info
- Why is value unusable?
  - Where did unusable value originate?

Origin Tracking Implementations

- Null pointers (Java)
  - Jikes RVM
- Undefined values (native code)
  - Valgrind’s MemCheck
Origin Tracking Implementations

- Null pointers (Java)
  - Jikes RVM  Jikes RVM Research Archive
- Undefined values (native code)
  - Valgrind’s MemCheck  Valgrind Source Code Repository

Identifies origin for 32-bit values
- 47 of 147 are 32-bit
- For 32-bit: 34 of 47 identified

Adds negligible overhead (28X → 28X)
Storing Origins in Null Values

- **Requirements**
  - Need bits in null values → multiple null values
  - Program operations support null values
  - Recover origin at exception

Null ⇔ high 5 bits are zero (27 bits available)
Reserve & protect address range: 0x00000000–0x07fffffff

<table>
<thead>
<tr>
<th>00000₂</th>
<th>Method ID</th>
<th>Bytecode index</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 bits</td>
<td>14 bits</td>
<td>13 bits</td>
</tr>
</tbody>
</table>
### Implementing Java Operations

<table>
<thead>
<tr>
<th></th>
<th>Java semantics</th>
<th>Standard VM</th>
<th>Origin tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment of null constant</td>
<td>obj = null;</td>
<td>obj = 0;</td>
<td>obj = this_location;</td>
</tr>
<tr>
<td>Object allocation</td>
<td>obj = new Object();</td>
<td>foreach ref slot i obj[i] = 0;</td>
<td>foreach ref slot i obj[i] = this_location;</td>
</tr>
<tr>
<td>Null reference comparison</td>
<td>if (obj == null) {</td>
<td>if (obj == 0) {</td>
<td>if (obj &amp; 0xf8000000) == 0) {</td>
</tr>
<tr>
<td>General reference comparison</td>
<td>if (obj1 == obj2) {</td>
<td>if (obj1 == obj2) {</td>
<td>if (((obj1 &amp; 0xf8000000) == 0) ? (obj2 &amp; 0xf8000000) == 0) : (obj1 == obj2)) {</td>
</tr>
</tbody>
</table>

Object allocation
- `obj = new Object();`
- `foreach ref slot i obj[i] = 0;`
- `foreach ref slot i obj[i] = this_location;`

Null reference comparison
- `if (obj == null) {`
- `if (obj == 0) {`
- `if (obj & 0xf8000000) == 0) {`

General reference comparison
- `if (obj1 == obj2) {`
- `if (obj1 == obj2) {`
- `if (((obj1 & 0xf8000000) == 0) ? (obj2 & 0xf8000000) == 0) : (obj1 == obj2)) {`
## Implementing Java Operations

<table>
<thead>
<tr>
<th></th>
<th>Java semantics</th>
<th>Standard VM</th>
<th>Origin tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assignment of null constant</strong></td>
<td><code>obj = null;</code></td>
<td><code>obj = 0;</code></td>
<td><code>obj = this_location;</code></td>
</tr>
<tr>
<td><strong>Object allocation</strong></td>
<td><code>obj = new Object();</code></td>
<td><code>for each ref slot i</code></td>
<td><code>for each ref slot i</code></td>
</tr>
<tr>
<td></td>
<td><code>obj[i] = 0;</code></td>
<td><code>obj[i] = this_location;</code></td>
<td><code>obj[i] = this_location;</code></td>
</tr>
<tr>
<td><strong>Null reference comparison</strong></td>
<td><code>if (obj == null) {</code></td>
<td><code>if (obj == 0) {</code></td>
<td><code>if (obj &amp; 0xf8000000) == 0) {</code></td>
</tr>
<tr>
<td></td>
<td><code>if (obj1 == obj2) {</code></td>
<td><code>if (obj1 == obj2) {</code></td>
<td><code>if (obj1 == obj2) {</code></td>
</tr>
<tr>
<td><strong>General reference comparison</strong></td>
<td><code>if (obj1 == obj2) {</code></td>
<td><code>if (obj1 == obj2) {</code></td>
<td><code>if (((obj1 &amp; 0xf8000000) == 0) ? ((obj2 &amp; 0xf8000000) == 0) : (obj1 == obj2)) {</code></td>
</tr>
</tbody>
</table>

---

The table provides implementations of various Java operations, comparing Java semantics with Standard VM and Origin tracking. Each operation is described with corresponding code snippets to illustrate how null is handled, objects are allocated, and null references are compared.
### Implementing Java Operations

<table>
<thead>
<tr>
<th>Java semantics</th>
<th>Standard VM</th>
<th>Origin tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assignment of null constant</strong></td>
<td>obj = null;</td>
<td>obj = 0;</td>
</tr>
<tr>
<td><strong>Object allocation</strong></td>
<td>obj = new Object();</td>
<td>foreach ref slot i, obj[i] = 0;</td>
</tr>
<tr>
<td><strong>Null reference comparison</strong></td>
<td>if (obj == null) {</td>
<td>if (obj == 0) {</td>
</tr>
<tr>
<td><strong>General reference comparison</strong></td>
<td>if (obj1 == obj2) {</td>
<td>if (obj1 == obj2) {</td>
</tr>
<tr>
<td><strong>General assignment</strong></td>
<td>obj1 = obj2;</td>
<td>obj1 = obj2;</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Unusable values
- Instances of origin tracking
  - Null pointer exceptions (Java)
  - Undefined value errors (C/C++)
- Redefining program operations
- Evaluation
  - Performance
  - Usefulness
- Related Work

Methodology

- Adaptive methodology
  - Mix of application & compilation time
  - Single iteration; 25 trials
- DaCapo, SPEC JBB2000, SPEC JVM98
- 3.6 GHz Pentium 4 w/Linux
Performance of Java Implementation

Finding and Fixing Bugs

- 12 real NPEs from SourceForge

  Origin: identified by origin tracking?

  Triviality: origin obvious by inspection?

  Usefulness: origin useful for fixing bug?
### Null Pointer Exceptions

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
<th>Exception description</th>
<th>Origin?</th>
<th>Trivial?</th>
<th>Useful?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mckoi SQL DB</td>
<td>94,681</td>
<td>Access closed connection</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>FreeMarker</td>
<td>64,442</td>
<td>JUnit test crashes unexpectedly</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Plot without x-axis</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Invalid class name</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Malformed XML document</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>47,871</td>
<td>Empty default case</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>JODE</td>
<td>44,937</td>
<td>Exception decompiling class</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Use built-in class as variable</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Potentially useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Stacked XY plot with lines</td>
<td>Yes</td>
<td>Somewhat nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Problem accessing <strong>doc</strong></td>
<td>Yes</td>
<td>Somewhat nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Package and import on same line</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Close Eclipse while deleting project</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
</tbody>
</table>
## Null Pointer Exceptions

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
<th>Exception description</th>
<th>Origin?</th>
<th>Trivial?</th>
<th>Useful?</th>
</tr>
</thead>
<tbody>
<tr>
<td>McKoi SQL DB</td>
<td>94,681</td>
<td>Access closed connection</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>FreeMarker</td>
<td>64,442</td>
<td>JUnit test crashes unexpectedly</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Plot without x-axis</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Invalid class name</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Malformed XML document</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>47,871</td>
<td>Empty default case</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>JODE</td>
<td>44,937</td>
<td>Exception decompiling class</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Use built-in class as variable</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Potentially useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Stacked XY plot with lines</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Problem accessing <strong>doc</strong></td>
<td>Yes</td>
<td>Somewhat nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Package and import on same line</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Close Eclipse while deleting project</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
</tbody>
</table>
Null Pointer Exceptions

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
<th>Exception description</th>
<th>Origin?</th>
<th>Trivial?</th>
<th>Useful?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mckoi SQL DB</td>
<td>94,681</td>
<td>Access closed connection</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>FreeMarker</td>
<td>64,442</td>
<td>JUnit test crashes unexpectedly</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Plot without x-axis</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Invalid class name</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Definitely useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Malformed XML document</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>47,871</td>
<td>Empty default case</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>JOGE</td>
<td>44,937</td>
<td>Exception decompiling class</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Most likely useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Use built-in class as variable</td>
<td>Yes</td>
<td>Nontrivial</td>
<td>Potentially useful</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>223,869</td>
<td>Stacked XY plot with lines</td>
<td>Yes</td>
<td>Somewhat nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>Jython</td>
<td>144,739</td>
<td>Problem accessing <strong>doc</strong></td>
<td>Yes</td>
<td>Somewhat nontrivial</td>
<td>Marginally useful</td>
</tr>
<tr>
<td>JRefactory</td>
<td>231,338</td>
<td>Package and import on same line</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
<tr>
<td>Eclipse</td>
<td>2,425,709</td>
<td>Close Eclipse while deleting project</td>
<td>Yes</td>
<td>Trivial</td>
<td>Not useful</td>
</tr>
</tbody>
</table>

Debugging Timeline

- Early: Language design
- Static analysis
- Testing
- Deployment: Late
Debugging Timeline

Prevent bugs
- Memory bugs [Java & C#]
- Null pointer exceptions [Chalin & James ‘06]
  - Add “never null” types to Java
  - Programmer effort
  - Exceptions still possible
- Functional languages

Detect errors in **any** execution
- [FindBugs] [PMD] [ESC/Java] [JLint] [Metal]
- Dataflow analysis & pattern matching
- Program complexity ➔ conservative (false positives)
- Some intentionally unsound (false negatives)
Debugging Timeline

Catch errors in real executions
- Assertions
- Checking tools [Valgrind] [Purify]
- Dynamic slicing [Agrawal & Horgan ’90] [Zhang et al. ’07]

Powerful but high overhead

Ideal environment for debugging?
- Stack/dump reporting
- Invariant-based bug detection [Liblit et al. ’05]
  - Many executions
- Limited dynamic slicing
  [TaintCheck] [Chilimbi & Hauswirth ’04] [Origin tracking]
  - Single execution
  - Narrow focus
Summary

- Unusable values: tough bugs
  - Error stack trace not enough
  - Opportunity: store origin in place of unusable value
  - Managed and native implementations
- Java null origins: fast, useful, silent
- Add it to your VM today!

Thank you!
Extra slides

Runtime Overhead

- Simple checks + ref compare + field initialization
- Overhead (%)
- DaCapo, SPEC, All
- Origin tracking